Concepts of Operating System

Assignment 2

Part A

What will the following commands do?

* echo "Hello, World!"

- In this case, the command will display: Hello, World!

* name="Productive"

The command name="Productive" assigns the string Productive to a variable named name in Unix-like operating systems. -echo $name= Productive

* touch file.txt

-**Creating a New Empty File**

**- Updating Timestamps of an Existing File:**

* ls -a

**-** lists all files and directories in the current directory

* rm file.txt- is used to remove the file named file.txt from the current directory. This operation is permanent and does not move the file to a trash or recycle bin; once executed, the file is deleted and cannot be easily recovered.
* cp file1.txt file2.txt- **Copying Contents**: The cp command duplicates the content of file1.txt into file2.txt. If file2.txt doesn't exist, it will be created; if it does exist, it will be overwritten.
* mv file.txt /path/to/directory/

 **Moving a File**: If /path/to/directory/ is an existing directory, file.txt will be moved into that directory, retaining its original name.

 **Renaming a File**: If /path/to/directory/ is a new file name (not an existing directory), file.txt will be renamed to directory in the current directory.

* grep "pattern" file.txt- **Pattern Matching**: grep searches for the exact sequence of characters specified in the pattern within file.txt. It does not perform regular expression matching unless specified with the -E option.
* kill PID-The command kill PID in Unix-like operating systems sends a signal to the process identified by the Process ID (PID) specified. By default, it sends the SIGTERM (signal number 15), which requests the process to terminate gracefully.

mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt- The command sequence mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt performs the following operations:

1. **Create a Directory**: mkdir mydir creates a new directory named mydir in the current working directory.
2. **Change to the New Directory**: cd mydir changes the current working directory to mydir.
3. **Create an Empty File**: touch file.txt creates an empty file named file.txt within mydir.
4. **Write Text to the File**: echo "Hello, World!" > file.txt writes the string "Hello, World!" to file.txt, overwriting any existing content.
5. **Display the File's Content**: cat file.txt displays the content of file.txt, which is "Hello, World!".

* ls -l | grep ".txt"-The command sequence ls -l | grep ".txt" lists all files in the current directory with detailed information and then filters the output to display only those files whose names contain the substring .txt.
* cat file1.txt file2.txt | sort | uniq

 **Concatenate Files**: cat file1.txt file2.txt reads the contents of file1.txt and file2.txt sequentially and outputs them to the standard output.

 **Sort the Output**: The sort command takes the concatenated output and arranges the lines in lexicographical (alphabetical) order.

 **Remove Duplicate Lines**: The uniq command filters out adjacent duplicate lines from the sorted output, displaying only unique lines.

* ls -l | grep "^d"- ls -l This command lists all files and directories in the current directory, providing detailed information such as permissions, number of links, owner, group, size, and modification date.
* This command filters the output to display only lines that start with the character d, which indicates a directory in the ls -l output.
* grep -r "pattern" /path/to/directory/

 **Recursive Search**: The -r (or --recursive) option tells grep to read all files under each directory, recursively. This means it will search through all files in /path/to/directory/ and its subdirectories.

*  **Pattern Matching**: The "pattern" is the text or regular expression you want to search for within the files.
*  **Directory Specification**: /path/to/directory/ specifies the directory where the search begins.
* cat file1.txt file2.txt | sort | uniq –d
* cat file1.txt file2.txt

This command reads the contents of file1.txt and file2.txt sequentially and outputs them to the standard output.

sort

The sort command takes the concatenated output and arranges the lines in lexicographical (alphabetical) order.

* **Display Duplicate Lines**:
* bash
* Copy
* uniq -d
* The uniq command filters out adjacent duplicate lines from the sorted output. The -d option tells uniq to display only the lines that are repeated.
* **Display Duplicate Lines**:

uniq -d

* The uniq command filters out adjacent duplicate lines from the sorted output. The -d option tells uniq to display only the lines that are repeated.

chmod 644 file.txt- The command chmod 644 file.txt sets the permissions of file.txt to rw-r--r--. Here's a breakdown of what this means:

* **Owner Permissions (rw-)**:
  + **Read (r)**: The owner can read the contents of the file.
  + **Write (w)**: The owner can modify the contents of the file.
  + **No Execute (-)**: The owner cannot execute the file as a program.
* **Group Permissions (r--)**:
  + **Read (r)**: Members of the file's group can read the contents of the file.
  + **No Write (-)**: Group members cannot modify the contents of the file.
  + **No Execute (-)**: Group members cannot execute the file as a program.
* **Others Permissions (r--)**:
  + **Read (r)**: All other users can read the contents of the file.
  + **No Write (-)**: Others cannot modify the contents of the file.
  + **No Execute (-)**: Others cannot execute the file as a program.

In numeric mode, permissions are represented by a three-digit octal number, where each digit corresponds to the permissions for the owner, group, and others, respectively. The digits are calculated by adding the values for read (4), write (2), and execute (1) permissions:

* **Owner (rw-)**: Read (4) + Write (2) = 6
* **Group (r--)**: Read (4) = 4
* **Others (r--)**: Read (4) = 4

Therefore, chmod 644 file.txt sets the permissions to rw-r--r--.

This permission setting is commonly used for files that should be readable and writable by the owner, but only readable by others.

* cp -r source\_directory destination\_directory -

The command cp -r source\_directory destination\_directory recursively copies the contents of source\_directory into destination\_directory. Here's a breakdown of the command:

* **cp**: The cp command is used to copy files and directories in Unix-like operating systems.
* **-r**: The -r (or --recursive) option tells cp to copy directories and their contents recursively. This means it will copy all files and subdirectories within source\_directory to destination\_directory.
* **source\_directory**: This is the path to the directory whose contents you want to copy.
* **destination\_directory**: This is the path to the directory where you want to copy the contents of source\_directory.
* find /path/to/search -name "\*.txt"

he command find /path/to/search -name "\*.txt" searches for all files with a .txt extension within the specified directory (/path/to/search) and its subdirectories. Here's a breakdown of the command:

1. **find**: The find command is used to search for files and directories in a directory hierarchy based on various criteria.
2. **/path/to/search**: This specifies the directory where the search begins. Replace /path/to/search with the actual path where you want to start the search.
3. **-name "\*.txt"**: This option tells find to look for files whose names match the pattern \*.txt, which includes all files ending with the .txt extension.

**Example**:

If you want to find all .txt files in the /home/user/docs/ directory and its subdirectories, you would run:

find /home/user/docs/ -name "\*.txt"

This command will list all .txt files within /home/user/docs/ and its subdirectories.

**Additional Options**:

* To perform a case-insensitive search:

bash

Copy

find /home/user/docs/ -iname "\*.txt"

* To execute a command on each found file (e.g., to display the contents of each .txt file):

find /home/user/docs/ -name "\*.txt" -exec cat {} \;

This command will display the contents of each .txt file found.

chmod u+x file.txt- The command chmod u+x file.txt grants the execute permission to the user (owner) of the file file.txt. Here's a breakdown of the command:

* **chmod**: The chmod command is used to change the permissions of a file or directory in Unix-like operating systems.
* **u**: Represents the user (owner) of the file.
* **+x**: Adds the execute permission.
* **file.txt**: The target file whose permissions are being modified.

By executing this command, the owner of file.txt will have the ability to execute the file as a program, provided the file contains executable code.

**Example**:

If file.txt is a script, running chmod u+x file.txt allows the owner to execute it directly from the command line.

**Note**:

Ensure that the file contains executable code and that you have the necessary permissions to modify the file's attributes.

For more detailed information on the chmod command and file permissions, you can refer to the [Linode guide on modifying file permissions with chmod](https://www.linode.com/docs/guides/modify-file-permissions-with-chmod/).

* echo $PATH- The command echo $PATH displays the current value of the PATH environment variable in Unix-like operating systems. The PATH variable is a colon-separated list of directories that the shell searches to find executable files. When you enter a command, the shell looks through these directories in order to locate the corresponding executable.

Part B

**Identify True or False:**

* ls is used to list files and directories in a directory-True.
* mv is used to move files and directories.- True
* cd is used to copy files and directories.- False.
* pwd stands for "print working directory" and displays the current directory.-True
* grep is used to search for patterns in files.-true
* chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.-True
* mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1if directory1 does not exist.-True
* rm -rf file.txt deletes a file forcefully without confirmation.-True
* Identify the Incorrect Commands:

Identify the Incorrect Commands:

* **chmodx is used to change file permissions.-**

The command chmodx is incorrect. The correct command to change file permissions is chmod. The chmod command in Unix-like operating systems is used to change the permissions of files and directories, allowing users to control who can read, write, or execute them.

* **cpy is used to copy files and directories.**

The command cpy is incorrect for copying files and directories in Unix-like operating systems. The correct command is cp. The cp command is used to copy files and directories.

* **mkfile is used to create a new file.**

The command mkfile is not used to create a new file in Unix-like operating systems. Instead, the touch command is commonly used to create an empty file.

* **catx is used to concatenate files.**

The command catx is incorrect for concatenating files. The correct command is cat. The cat command is used to concatenate and display the contents of files in Unix-like operating systems. cat file1.txt file2.txt

* rn is used to rename files.
* The command rn is incorrect for renaming files in Unix-like operating systems. The correct command is mv. The mv command is used to move or rename files and directories.

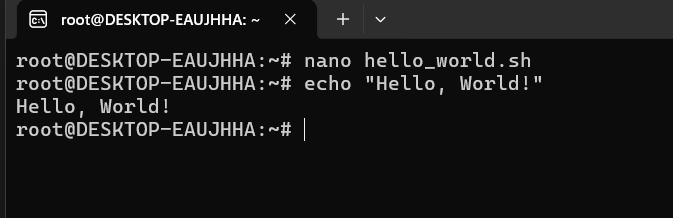
**Example**:

* To rename a file named oldname.txt to newname.txt, you would use:

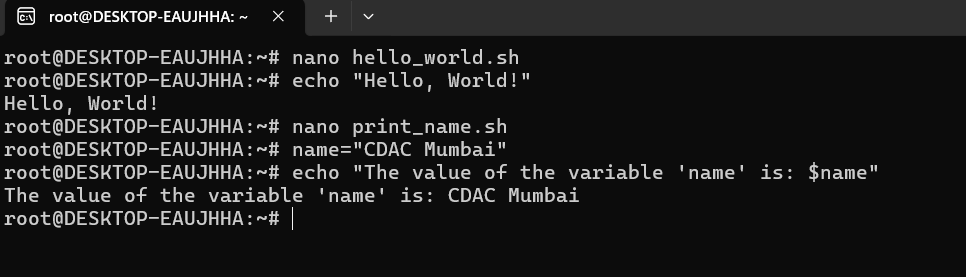
mv oldname.txt newname.txt

Part C

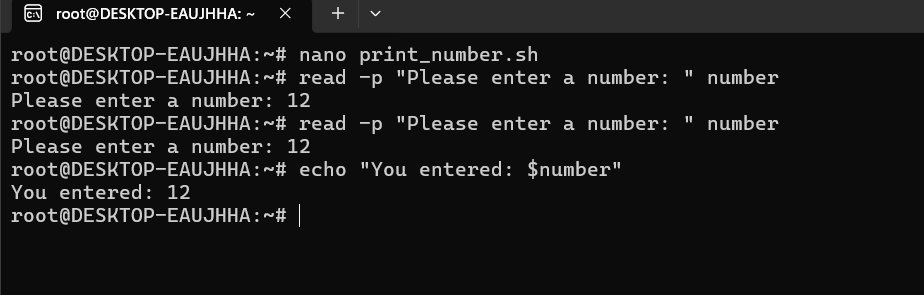
Question 1: Write a shell script that prints "Hello, World!" to the terminal.



Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.

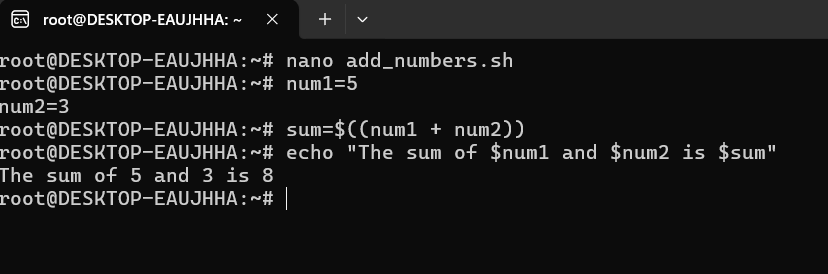


Question 3: Write a shell script that takes a number as input from the user and prints it.



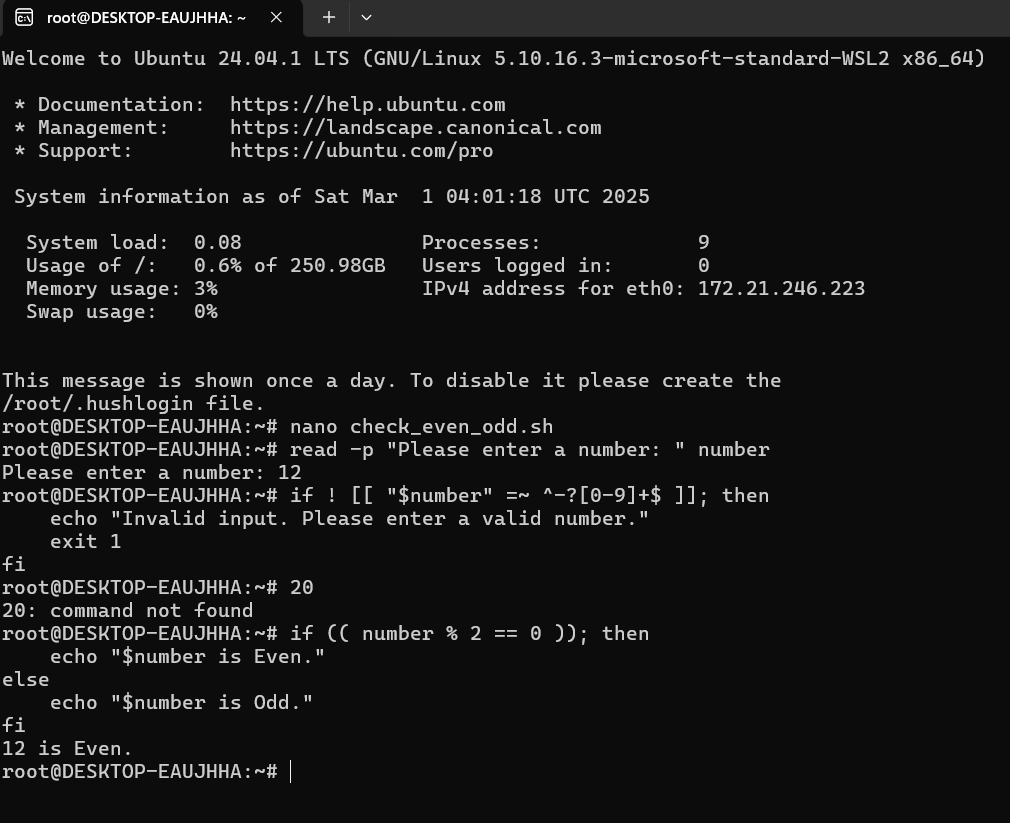
Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the

result.

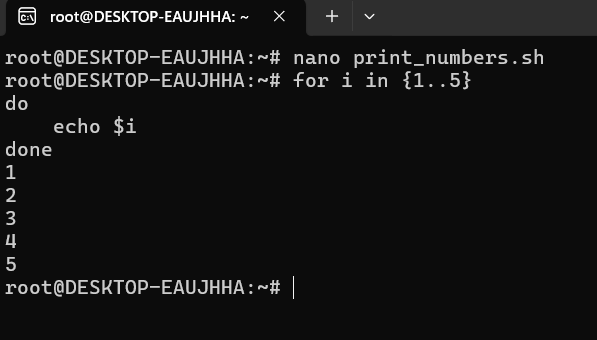


Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise

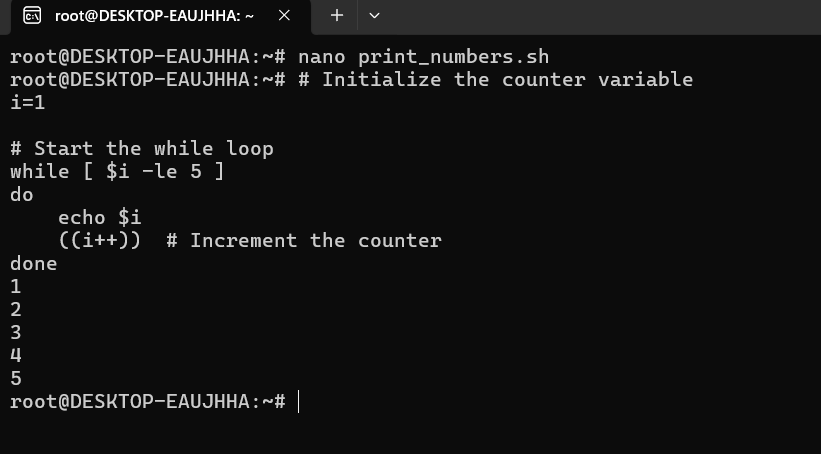
prints "Odd".



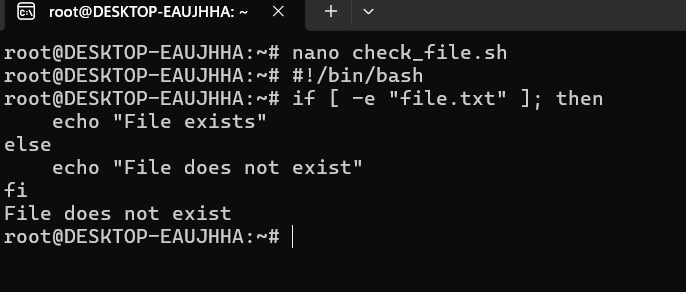
Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.



Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

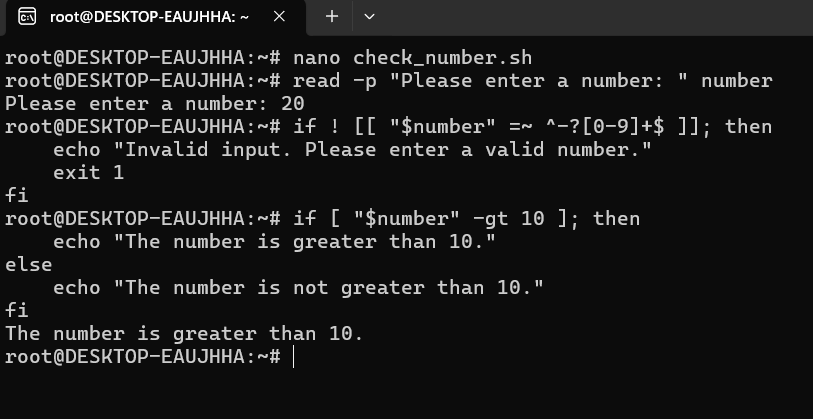


Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and

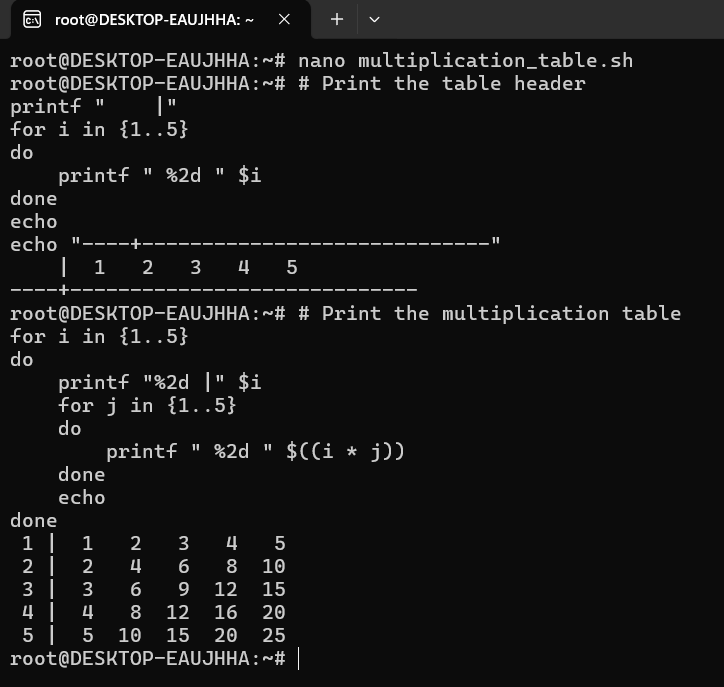
prints a message accordingly.



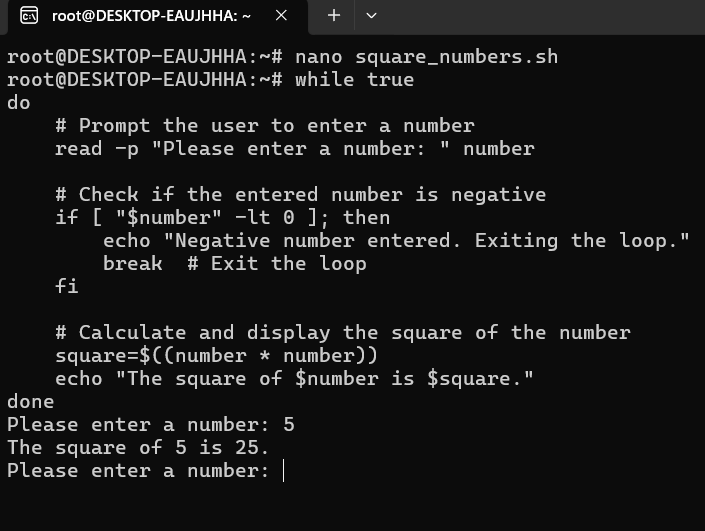
Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers

from 1 to 5. The output should be formatted nicely, with each row representing a number and each

column representing the multiplication result for that number.



Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.



Part E

**1. Consider the following processes with arrival times and burst times:**

**| Process | Arrival Time | Burst Time |**

**|---------|--------------|------------|**

**| P1 | 0 | 5 |**

**| P2 | 1 | 3 |**

**| P3 | 2 | 6 |**

**Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.**

| **Process** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** |
| --- | --- | --- | --- | --- | --- |
| P1 | 0 | 5 | 5 | 5 | 0 |
| P2 | 1 | 3 | 8 | 7 | 4 |
| P3 | 2 | 6 | 14 | 12 | 6 |

**Calculate Average Waiting Time.**

Average Waiting Time = (WT of P1 + WT of P2 + WT of P3) / Number of Processes

Average Waiting Time = (0 + 4 + 6) / 3

4.333333

(0 + 5 + 8) / 3=4.333333

**2. Consider the following processes with arrival times and burst times:**

**| Process | Arrival Time | Burst Time |**

**|---------|--------------|------------|**

**| P1 | 0 | 3 |**

**| P2 | 1 | 5 |**

**| P3 | 2 | 1 |**

**| P4 | 3 | 4 |**

**Calculate the average turnaround time using Shortest Job First (SJF) scheduling.**

| **Process** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** |
| --- | --- | --- | --- | --- | --- |
| P1 | 0 | 3 | 4 | 4 | 1 |
| P2 | 1 | 5 | 13 | 12 | 7 |
| P3 | 2 | 1 | 1 | 1 | 0 |
| P4 | 3 | 4 | 8 | 5 | 1 |

**Calculate Average Turnaround Time.**

Average Turnaround Time = (TAT of P1 + TAT of P2 + TAT of P3 + TAT of P4) / Number of Processes

Average Turnaround Time = (4 + 12 + 1 + 5) / 4

Average Turnaround Time = 22 / 4

Average Turnaround Time = 5.5 units

Therefore, the average turnaround time for the given processes using SJF scheduling is 5.5 units.

**3. Consider the following processes with arrival times, burst times, and priorities (lower number**

**indicates higher priority):**

**| Process | Arrival Time | Burst Time | Priority |**

**|---------|--------------|------------|----------|**

**| P1 | 0 | 6 | 3 |**

**| P2 | 1 | 4 | 1 |**

**| P3 | 2 | 7 | 4 |**

**| P4 | 3 | 2 | 2 |**

**Calculate the average waiting time using Priority Scheduling.**

| **Process** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** |
| --- | --- | --- | --- | --- | --- |
| P1 | 0 | 6 | 13 | 13 | 7 |
| P2 | 1 | 4 | 5 | 4 | 0 |
| P3 | 2 | 7 | 20 | 18 | 11 |
| P4 | 3 | 2 | 7 | 4 | 2 |

**Calculate Average Waiting Time.**

Average Waiting Time = (WT of P1 + WT of P2 + WT of P3 + WT of P4) / Number of Processes

Average Waiting Time = (7 + 0 + 11 + 2) / 4

Average Waiting Time = 20 / 4

Average Waiting Time = 5 units

Therefore, the average waiting time for the given processes using Priority Scheduling is 5 units.

**4. Consider the following processes with arrival times and burst times, and the time quantum for**

**Round Robin scheduling is 2 units:**

**| Process | Arrival Time | Burst Time |**

**|---------|--------------|------------|**

**| P1 | 0 | 4 |**

**| P2 | 1 | 5 |**

**| P3 | 2 | 2 |**

**| P4 | 3 | 3 |**

**Calculate the average turnaround time using Round Robin scheduling.**

| **Process** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** |
| --- | --- | --- | --- | --- | --- |
| P1 | 0 | 4 | 9 | 9 | 5 |
| P2 | 1 | 5 | 14 | 13 | 8 |
| P3 | 2 | 2 | 5 | 3 | 1 |
| P4 | 3 | 3 | 12 | 9 | 6 |

**Step 4: Calculate Average Turnaround Time**

Average Turnaround Time = (TAT of P1 + TAT of P2 + TAT of P3 + TAT of P4) / Number of Processes

Average Turnaround Time = (9 + 13 + 3 + 9) / 4 = 34 / 4 = 8.5 units

Therefore, the average turnaround time for the given processes using Round Robin scheduling with a 2-unit time quantum is 8.5 units.

**5. Consider a program that uses the fork() system call to create a child process. Initially, the parent**

**process has a variable x with a value of 5. After forking, both the parent and child processes**

**increment the value of x by 1.**

**What will be the final values of x in the parent and child processes after the fork() call?**

In a program that uses the fork() system call to create a child process, the child receives a copy of the parent's address space, including variables. This means that changes made to a variable in the child process do not affect the parent process's variable, and vice versa.

**Initial Setup:**

* Parent process variable x is initialized to 5.

**After fork() is called:**

* Both parent and child processes have their own separate copies of x, each initialized to 5.

**In the Parent Process:**

* The parent increments its own x by 1, changing its value to 6.

**In the Child Process:**

* The child increments its own x by 1, changing its value to 6.

**Conclusion:**

* The parent process's x is 6.
* The child process's x is also 6.

These values are independent because the fork() system call creates a new process with a separate memory space. Therefore, modifications to variables in one process do not affect the corresponding variables in the other process.